1

219800-13-T(2)

Technical Report

MaxVideo NEIGHBORHOOD PROCESSOR PIPELINE BOARD OPERATION SOFTWARE DESCRIPTIONS

J. JONIK

JULY 1990

Prepared for:
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-5800

Contract No. 42-3638

TECHNICAL REPORT STANDARD TITLE PAGE

1. Report No. 219800-13-T(2)	2. Government Accession No.	3. Recipient's Catalog No.

4. Title and Subtitle
MaxVideo Neighborhood Processor Pipeline Board Operation Software Descriptions

5. Report Date
July 1990

6. Performing Organization Code
ERIM

7. Author(s)
J. Jonik

8. Performing Organization Report No.
219800-13-T(2)

9. Performing Organization Name and Address
Environmental Research Institute of Michigan
P.O. Box 8618
Ann Arbor, MI 48107-8618

10. Work Unit No.
PTD-90-063

11. Contract or Grant No.
42-3638

12. Sponsoring Agency Name and Address
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-5800

13. Type of Report and Period Covered
Software Descriptions
Sept. 1989-Oct. 1990

14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This document contains a description of the subroutines that form the MVNPP Operation Software. The subroutines have been written to provide software support to program, operate, and test the MVNPP in a Datacube environment. Provided routines include reading and writing to all MVNPP registers, reading stagecode from a file and programming the stages, sizing the pipeline, computing the delay added from the pipeline, and executing stage readback memory and internal registers commands.

The code has been written to comply with Sandia's Datacube Software Requirements and Guidelines document. Makefiles have been provided to build the libraries on the Sun OS. Calling procedures and descriptions of parameters and return values are described for each routine. Keywords:
Software loading from tape.

17. Key Words
Software

18. Distribution Statement
Approved for public release; distribution is unlimited.

19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page)	21. No. of Pages 34	22. Price

CONTENTS

1.0 INTRODUCTION

This document contains a description of the subroutines that form the MaxVideo (tm) Neighborhood Processor Pipeline (MVNPP) Operation Software. The software has been designed to operate under Sun OS, Release 4.0.3, on a Sun 3 Workstation. The modules are divided into three subdirectories: 'toplevel', 'lowlevel', and 'reglevel', similar to the routines in maxtools. Each subdirectory has its own Makefile that adds the code to the 'mvnpplib' library. The subdirectories are located under 'maxtools/mvnpp'.

The code has been written to comply with Sandia's Datacube (tm) Software Requirements and Guidelines document. Two new files have been added to the maxtools include directory: mvnppbits.h and mvnppStageop.h. The second include file is only used when converting a C4PL (tm) stagecode file to the format used by the MVNPP software routines; this is explained later in the document.

A new directory, 'mvnpptest', has been added to maxtools/mains. 'mvnpptest' contains the following subdirectories: 'clbusprog', 'demo', 'gonogo', 'imagecirc', and 'makelnoc'. See the MVNPP High-Level Test Software Specification (IPTL-90-089) for a description of the programs in the listed directories.

A new module was added to the 'maxtools/roimath' directory that provides access to the utility from a subroutine call. 'getROIparams' is a modified 'roimaster' routine that calls a new version of 'roitiming' (the new version is called 'roitime2'). New versions of the existing 'roimath' modules were created so that no messages are output at run time. The object code is put into 'roimathlib'. The end of this document contains a description of the calling procedure for 'getROIparams'.

The reader is referred to the Stage Programmer's Manual (IPTL-89-294) and the MVNPP Programmer's Manual (IPTL-90-014) for information regarding the purpose or function of the routines listed in this document, i.e., register descriptions, stage operations, etc.

MaxVideo and Datacube are registered trademarks of Datacube Incorporated.
C4PL is a registered trademark of the Environmental Research Institute of Michigan.

2.0 Loading Software from Tape

The following directories and files are on the tape provided:

```
mains/mvnpptest/imagecirc
mains/mvnpptest/clbusprog
mains/mvnpptest/gonogo
mains/mvnpptest/makelnoc
mains/mvnpptest/demo

mains/vmetest/regrw
mains/vmetest/pipesize
mains/vmetest/fifoerrors
mains/vmetest/utils

mvnpp/reglevel
mvnpp/lowlevel
mvnpp/toplevel

roimath/getROIparams.c
roimath/roitime2.c
roimath/Makemathlib

include/mvnppbits.h
include/mvnppstageop.h
```

Use the following commands to extract the files:

```
cd maxtools

tar xvf /dev/rst8
```

Each of the 'mvnpp' subdirectories has a Makefile that will compile the routines and add them to the mvnpp library. The file 'MakeROIlib' file in the roimath directory creates a library for the roimath utility subroutines. Enter 'make -f MakeROIlib sun' to build this library.

3.0 MVNPP Software Reglevel Routines

The routines in the 'reglevel' directory provide a means to read from and write to the MVNPP registers. As stated above, refer to the MVNPP Programmer's Manual for a complete description of the register layout and functions. The open and close routines for the MVNPP are also located in this directory.

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppOpen
 *
 *    Allocate virtual memory for MVNPP board control registers.
 *
 *-----------------------------------------------------------------------*/

 XX_DEV       *mvnppOpen(mvnpprbnum, mvnpprbase, mvnpprmode);
      int mvnpprbnum;
      ULONG mvnpprbase;
      int mvnpprmode;

      mvnpprbnum - board index
      mvnpprbase - register base address [0 | 1 | 2 | 3]
      mvnpprmode - addressing mode    [USE_A24 | USE_A32]

      Return:  XX_DEV structure containing the virtual base address
               for the MVNPP registers.

--------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppClose
 *
 *    Free virtual memory allocated for the MVNPP board control registers.
 *
 *-----------------------------------------------------------------------*/

 void  mvnppClose(mvnpp);
      XX_DEV *mvnpp;

--------------------------------------------------------------------------
```

```
/*---------------------------------------------------------------------*
 *
 *    mvnpprSR
 *
 *    Return the contents of the status register; the one unused bit
 *    is not masked off.
 *
 *---------------------------------------------------------------------*/

int  mvnpprSR(mvnpp);
     XX_DEV *mvnpp;

     Return:  contents of status register


     The bit definitions of the status register as found in the
     mvnppbits.h file are as follows:

     MVNPP_SR_PIPE_ERR                0x8000  /* Pipeline error flag */
     MVNPP_SR_FIFO_OVERRUN            0x4000  /* FIFO Overrun error flag */
     MVNPP_SR_FIFO_UNDERRUN           0x2000  /* FIFO Underrun error flag */
     MVNPP_SR_PIPE_COLLISION          0x1000  /* Pipe collision error flag */
     MVNPP_SR_CNSW                    0x0400  /* ROI output configuration
                                                   wire bit */
     MVNPP_SR_CNSE                    0x0200  /* ROI output configuration
                                                   sense enable bit */
     MVNPP_SR_IMAG_XFORMD             0x0100  /* Image transformed flag */

     MVNPP_SR_BD_ID_MSK               0x00ff  /* board ID code mask */
-----------------------------------------------------------------------------
```

```
/*------------------------------------------------------------------*
 *
 *    mvnpprCR
 *
 *    Return the contents of the control register.  Bits 11 - 15 are
 *    unused, and bits 0 - 2 are write only; these bits are masked
 *    off.
 *
 *------------------------------------------------------------------*/

int  mvnpprCR(mvnpp);
     XX_DEV *mvnpp;
```

Return - contents of control register

The bit definitions of the control register readable bits as found in the mvnppbits.h file are as follows:

```
MVNPP_CR_CL_HI_ALT              0x0200  /* CL transfer hi alternate  */
MVNPP_CR_CL_LO_ALT              0x0100  /* CL transfer low alternate */
MVNPP_CR_ROI_ALT                0x0080  /* ROI transfer alternate    */

MVNPP_CR_CL_HI_PRI              0x0000  /* CL transfer hi primary  */
MVNPP_CR_CL_LO_PRI              0x0000  /* CL transfer low primary */
MVNPP_CR_ROI_PRI                0x0000  /* ROI transfer primary    */

MVNPP_CR_ROI_TIM0               0x0000  /* ROI timing bus 0 */
MVNPP_CR_ROI_TIM1               0x0020  /* ROI timing bus 1 */
MVNPP_CR_ROI_TIM2               0x0040  /* ROI timing bus 2 */
MVNPP_CR_ROI_TIM3               0x0060  /* ROI timing bus 3 */

MVNPP_CR_HSS_CYTO               0x0000  /* Hi speed I/O select options *
MVNPP_CR_HSS_ROI                0x0008
MVNPP_CR_HSS_CL                 0x0010
MVNPP_CR_HSS_VME                0x0018
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppwCR
 *
 *    Write the given value to the control register.
 *
 *-----------------------------------------------------------------------*/

 void mvnppwCR(mvnpp, data);
      XX_DEV *mvnpp;
      int data;
```

The bit definitions of the control register as tound in the mvnppbits.h file are as follows:

```
MVNPP_CR_CL_HI_ALT                0x0200  /* CL transfer hi alternate  */
MVNPP_CR_CL_LO_ALT                0x0100  /* CL transfer low alternate */
MVNPP_CR_ROI_ALT                  0x0080  /* ROI transfer alternate    */

MVNPP_CR_CL_HI_PRI                0x0000  /* CL transfer hi primary  */
MVNPP_CR_CL_LO_PRI                0x0000  /* CL transfer low primary */
MVNPP_CR_ROI_PRI                  0x0000  /* ROI transfer primary    */

MVNPP_CR_ROI_TIM0                 0x0000  /* ROI timing bus 0 */
MVNPP_CR_ROI_TIM1                 0x0020  /* ROI timing bus 1 */
MVNPP_CR_ROI_TIM2                 0x0040  /* ROI timing bus 2 */
MVNPP_CR_ROI_TIM3                 0x0060  /* ROI timing bus 3 */

MVNPP_CR_HSS_CYTO                 0x0000  /* Hi speed I/O select options *
MVNPP_CR_HSS_ROI                  0x0008
MVNPP_CR_HSS_CL                   0x0010
MVNPP_CR_HSS_VME                  0x0018

MVNPP_CR_CLR_ERRORS               0x0004
MVNPP_CR_AUTO_FLUSH               0x0002
MVNPP_CR_IMAGE_START              0x0001
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnpprLineLen
 *
 *    Read the value of the line length count register; return the
 *    two's complement value of what was read.  This value is the number
 *    of pixels per line of the image to be transferred.
 *
 *-----------------------------------------------------------------------*/

int mvnpprLineLen(mvnpp)
     XX_DEV *mvnpp;

     Return - two's complement of line length count read from the register
```

```
-------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppwLineLen
 *
 *    Write the two's complement of the given value to the line length
 *    count register.  This value is the number of pixels per line of
 *    the image to be transferred.
 *
 *-----------------------------------------------------------------------*/

void mvnppwLineLen(mvnpp,length)
     XX_DEV *mvnpp;
     int length;     /* value to write after forming two's complement */
```

```
-------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnpprPLP
 *
 *    Read the value of the pipeline programming register.
 *
 *-----------------------------------------------------------------------*/

 USHORT mvnpprPLP(mvnpp)
      XX_DEV *mvnpp;

      Return - the contents of the pipeline programming register
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppwPLP
 *
 *    Write the value sent as a parameter to the pipeline programming
 *    register.  Note that only a 16-bit value may be written to this
 *    register.  The value written includes the LIS~, CIS~ and
 *    DIV~ signal states.
 *
 *-----------------------------------------------------------------------*/

 void mvnppwPipeLineProg(mvnpp,pipe_prog)
      XX_DEV *mvnpp;
      int  pipe_prog;   /* 16-bit value to write to the register */
```

```
/*----------------------------------------------------------------------*
 *
 *    mvnpprPID
 *
 *    Read the value of the pipeline image data register.  Return the
 *    long word containing the value.
 *
 *----------------------------------------------------------------------*/

 ULONG mvnpprPipeImageData(mvnpp)
      XX_DEV *mvnpp;

      Return - 32-bit value read from the register
```

```
-----------------------------------------------------------------------
```

```
/*----------------------------------------------------------------------*
 *
 *    mvnppwPID
 *
 *    Write the pipeline image data to the register.  The input parameters
 *    specify whether a byte, word, or long word is to be written along
 *    with the offset into the register.  For example, writing a byte to
 *    the 3rd byte of the register would have an offset of 2.  Valid
 *    offsets for bytes are 0 - 3, words may have an offset of 0 or 2,
 *    and longwords only an offset of 0.  For word and byte accesses,
 *    any of the valid offsets used produces the same results.
 *
 *----------------------------------------------------------------------*/

 void mvnppwPipeImageData(mvnpp,value,access,offset)
      XX_DEV *mvnpp;
      int     value   /* value to write to the register */
      int     access  /* BYTE_ACCESS, WORD_ACCESS, LWORD_ACCESS */
      int     offset  /* 0 - 3 */
```

```
-----------------------------------------------------------------------
```

```
/*----------------------------------------------------------------------*
 *
 *    mvnpprIHS
 *
 *    Read the value of the image HStart count register; take the two's
 *    complement of the value and subtract 1 before returning.
 *
 *----------------------------------------------------------------------*/

int mvnpprIHS(mvnpp)
     XX_DEV *mvnpp;

     Return - the two's complement minus 1 of the value read from the
              HStart count register


------------------------------------------------------------------------
```

```
/*----------------------------------------------------------------------*
 *
 *    mvnppwIHS
 *
 *    Take the two's complement of (hoffset + 1) and write
 *    to the image Hstart count register.
 *
 *----------------------------------------------------------------------*/

void mvnppwIHS(mvnpp,hoffset)
     XX_DEV *mvnpp;
     int     hoffset;


------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnpprIHL
 *
 *    Read the value of the image HLength count register; take the two's
 *    complement of the value before returning.
 *
 *-----------------------------------------------------------------------*/

int mvnpprIHL(mvnpp)
     XX_DEV *mvnpp;

     Return - the two's complement of the value read from the HLength count
              register

 ---------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppwIHL
 *
 *    Take the two's complement of line_length and write to the
 *    image Hlength count register.
 *
 *-----------------------------------------------------------------------*/

void mvnppwIHL(mvnpp,line_length)
     XX_DEV *mvnpp;
     int     line_length;

 ---------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnpprIVS
 *
 *    Read the value of the image VStart count register; take the two's
 *    complement of the value and subtract 1 before returning.
 *
 *-----------------------------------------------------------------------*/

int mvnpprIVS(mvnpp)
     XX_DEV *mvnpp;

     Return - the two's complement minus 1 of the value read from the
              VStart count register

 --------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *
 *    mvnppwIVS
 *
 *    Take the two's complement of (num_hsyncs + 1) and write
 *    to the image Vstart count register.
 *
 *-----------------------------------------------------------------------*/

void mvnppwIVS(mvnpp,num_hsyncs)
     XX_DEV *mvnpp;
     int     num_hsyncs;

--------------------------------------------------------------------------
```

```
/*-------------------------------------------------------------------------*
 *
 *    mvnpprIVL
 *
 *    Return two's complement value of the image VLength count register.
 *
 *-------------------------------------------------------------------------*/

int mvnpprIVL(mvnpp)
     XX_DEV *mvnpp;

     Return - the two's complement of the value read from the VLength count
              register
```

```
/*-------------------------------------------------------------------------*
 *
 *    mvnppwIVL
 *
 *    Take the two's complement of the num_lines and write it
 *    to the image VLength count register.
 *
 *-------------------------------------------------------------------------*/

void mvnppwIVL(mvnpp,num_lines)
     XX_DEV *mvnpp;
     int    num_lines;
```

```
/*--------------------------------------------------------------------------*
 *
 *    mvnpprCLHS
 *
 *    Read the value of the image HStart count register; take the two's
 *    complement of the value and subtract 1 before returning.
 *
 *--------------------------------------------------------------------------*/

 int mvnpprCLHS(mvnpp)
      XX_DEV *mvnpp;

      Return - the two's complement minus 1 of the value read from the
               HStart count register


 ---------------------------------------------------------------------------




/*--------------------------------------------------------------------------*
 *
 *    mvnppwCLHS
 *
 *    Take the two's complement of (hoffset + 1) and write
 *    to the image Hstart count register.
 *
 *--------------------------------------------------------------------------*/

 void mvnppwCLHS(mvnpp,hoffset)
      XX_DEV *mvnpp;
      int     hoffset;


 --- -----------------------------------------------------------------------
```

```
/*---------------------------------------------------------------------------*
 *
 *    mvnpprCLHL
 *
 *    Read the value of the image HLength count register; take the two's
 *    complement of the value before returning.
 *
 *---------------------------------------------------------------------------*/

 int mvnpprCLHL(mvnpp)
      XX_DEV *mvnpp;

      Return - the two's complement of the value read from the HLength count
               register


 ---------------------------------------------------------------------------
```

```
/*---------------------------------------------------------------------------*
 *
 *    mvnppwCLHL
 *
 *    Take the two's complement of the line length sent and write to the
 *    image Hlength count register.
 *
 *---------------------------------------------------------------------------*/

 void mvnppwCLHL(mvnpp,line_length)
      XX_DEV *mvnpp;
      int     line_length;


 ---------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*/
 *
 *    mvnpprCLVS
 *
 *    Read the value of the image VStart count register; take the two's
 *    complement of the value and subtract 1 before returning.
 *
 *-----------------------------------------------------------------------*/

int mvnpprCLVS(mvnpp)
     XX_DEV *mvnpp;

     Return - the two's complement minus 1 of the value read from the
              VStart count register

---------------------------------------------------------------------------
```

```
/*----------------------------------------------------------------------*
 *
 *    mvnppwCLVS
 *
 *    Take the two's complement of (num_hsyncs + 1) and write
 *    to the image Vstart count register.
 *
 *-----------------------------------------------------------------------*/

void mvnppwCLVS(mvnpp,num_hsyncs)
     XX_DEV *mvnpp;
     int     num_hsyncs;

---------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------*
 *    mvnpprCLVL
 *
 *    Read the value of the image VLength count register; take the two's
 *    complement of the value before returning.
 *
 *----------------------------------------------------------------------*/

int mvnpprCLVL(mvnpp)
      XX_DEV *mvnpp;

      Return - two's complement of the value read from the VLength count
               register

-------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
 *    mvnppwCLVL
 *
 *    Take the two's complement of num_lines and write
 *    to the image VLength count register.
 *
 *----------------------------------------------------------------------*/

void mvnppwCLVL(mvnpp,num_lines)
      XX_DEV *mvnpp;
      int     num_lines;

-------------------------------------------------------------------------
```

4.0 MVNPP Software Lowlevel Routines

The 'lowlevel' directory provides routines to compute the pipeline delay added by the MVNPP, perform read-backs in order to retrieve the contents of the stage chip memory and registers, read in stagecode from an .LNOC file, and size the pipeline. Also included are several routines that are called to produce a buffer containing commands and stagecode to program the stage pipeline.

```
/*----------------------------------------------------------------------
/*   mvnppCL
/*
/*   Program the CL data transfer timing control registers of the
/*   MVNPP board.  This routine calls the bitlevel routines to
/*   program the CL bus hstart, hlength, vstart, and vlength count
/*   registers.
/*
/*---------------------------------------------------------------------*/

void mvnppCL(mvnpp, hstart, hlength, vstart, vlength)
      XX_DEV *mvnpp;
      USHORT hstart;
      USHORT hlength;
      USHORT vstart;
      USHORT vlength;
```

```
/*-----------------------------------------------------------------------*/
/*  mvnppDelay
/*
/*  Calculate the image delay introduced by the MVNPP board.  The delay
/*  varies based on the number of available stages, the number of
/*  active stages, and the line length of the image to transfer.  The
/*  delays are computed as follows:
/*
/*    latency = nactive * (line_length + 17) + (2 * (nstages - nactive))
/*
/*    vert_delay = latency / line_length
/*                    (+ 1 if latency % line_length != 0)
/*
/*    horiz_delay = 2
/*
/*  Note:  mvnppSizePipe returns nstages, and mvnppReadCode returns
/*      nactive.
/*
/*-----------------------------------------------------------------------*/

void mvnppDelay(nstages, nactive, line_length, vert_delay, horiz_delay)
      int nstages;
      int nactive;
      int line_length;
      int *vert_delay;
      int *horiz_delay;

      nstages - number of physical stages available
      nactive - number of active stages
      line_length - length of line in image
      vert_delay  - address to store vertical delay
      horiz_delay - address to store horizontal delay

---------------------------------------------------------------------------
```

```
/*------------------------------------------------------------------------
/*  mvnppFrameBeg
/*
/*  Put a global activate command in the given memory buffer.  The NULL
/*  that must follow commands is automatically included, and the number
/*  of words added to the buffer (2) is returned.  The mvnppLoadCmds
/*  routine calls this module.
/*
/*------------------------------------------------------------------------*/

int mvnppFrameBeg(memptr)
      USHORT *memptr;


      Return - the number of words written to the buffer - 2
```

```
--------------------------------------------------------------------------
```

```
/*------------------------------------------------------------------------
/*  mvnppFrameEnd
/*
/*  Issue a local deactivate command to each idle stage.  If a stage has
/*  been programmed, it will be in local ignore state, so it will not be
/*  deactivated.  Any extra deactivate commands do no harm.  Two NULLs
/*  are then sent for each stage in order to flush the pipeline.  A
/*  global image start command is then added to the buffer.
/*  The value returned is the number of words that were put in the
/*  buffer.  The mvnppLoadCmds module calls this routine.
/*
/*------------------------------------------------------------------------*/

int mvnppFrameEnd(nstages,memptr)
      int nstages;
      USHORT *memptr;


      Return - number of words added to the buffer
               (nstages * 4 + 2)
```

```
--------------------------------------------------------------------------
```

```
/*-------------------------------------------------------------------------------
 *  mvnppLReadInt
 *
 *  This module performs a local read-back of a stage's registers.  The first
 *  idle stage will receive a local read-back memory command followed by a
 *  NULL.  A number of valid pixels (~DIV active) and NULLs will follow.
 *  The pipeline programming register will be read between each command write
 *  and the results stored.  The 4-bit registers have the upper bits
 *  masked off.  See the MVNPP Stage Programmer's manual for a description
 *  of how the read-back works.
 *
 *-----------------------------------------------------------------------------*/

void mvnppLReadInt(mvnpp,nstages,reg)
      XX_DEV *mvnpp;
      int nstages;
      UCHAR *reg;

      nstages - number of stages available; used to determine the
                pipeline latency involved
      reg     - buffer to store the read back registers

------------------------------------------------------------------------------


/*-----------------------------------------------------------------------------
 *  mvnppLReadMem
 *
 *  Perform a local read-back memory command and store the memory contents
 *  in the array sent as a parameter.  The first stage in the idle state
 *  receives the command.  See the MVNPP Stage Programmer's Manual for a
 *  description of how the read-back works.
 *
 *-----------------------------------------------------------------------------*/

void mvnppLReadMem(mvnpp,mem_size,nstages,mem)
      XX_DEV *mvnpp;
      int mem_size;
      int nstages;
      UCHAR *mem;

      mem_size - number of bytes to read back from stage memory; the entire
                 PRAM and NRAM need not be read
      nstages - number of stages available; used to determine the
                pipeline latency involved
      mem     - buffer to store the read back memory

-------------------------------------------------------------------------------
```

```
/*--------------------------------------------------------------------------
/*  mvnppROI.c
/*
/*  Program the ROI image transfer timing control registers of the
/*  ERIM MVNPP board.  This routine calls the bitlevel routines to
/*  program the ROI bus hstart, hlength, vstart, and vlength count
/*  registers.
/*
/*--------------------------------------------------------------------------*/

void mvnppROI(mvnpp, hstart, hlength, vstart, vlength)
      XX_DEV *mvnpp;
      USHORT hstart;
      USHORT hlength;
      USHORT vstart;
      USHORT vlength;


----------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
 *
 *  mvnppReadCode
 *
 *  Read stage code from a file and put it in the stage program buffer.
 *
 *    The format of the stage code file is assumed to
 *    be the following...
 *
 *    <magic # = 0x0B>                char
 *    <dummy>                         char
 *    <number of stages (n)>          short
 *    <offset to stage code block 1>  int
 *            .
 *            .
 *    <offset to stage code block n>  int
 *    <size 1>                        short
 *    <stage code block 1>            char[size 1]
 *            .
 *            .
 *    <size n>                        short
 *    <stage code block n>            char[size n]
 *
 *  Stagecode created from C4PL and saved with the 'STORECODE' command may
 *  be converted to this format by running 'makeLnoc' externally.
 *  ('makeLnoc' is a main program located under 'mvnpptest/makelnoc'.)
 *  The conversion may also be done by calling 'mvnppCnvtOp' from within a
 *  program (a toplevel routine).
 *
 *  The stagecode buffer is filled with enough code to program the available
 *  stages, and the 'codesizes' array is updated with the size of each
 *  stage's program.  'nactive' is updated with the number of stages that
 *  code is available for; in most cases, this will be the number of
 *  active stages.
/*----------------------------------------------------------------------*/

void mvnppReadCode(fd, nstages, nactive, stagecode, codesizes)
      int fd;
      int nstages;
      int *nactive;
      UCHAR *stagecode;
      int *codesizes;

      nstages   - number of stages in the pipeline
      nactive   - number of active stages in the pipeline
      stagecode - buffer to hold the stage code
      codesizes - array of size nactive, containing the size of
                  the stage code programs for each stage
```

```
/*-----------------------------------------------------------------------
/* mvnppSizePipe
/*
/* Determine the number of stages in the pipeline.
/*
/*-----------------------------------------------------------------------*/

int mvnppSizePipe(mvnpp, maxstages)
      XX_DEV *mvnpp;
      int maxstages;


      Return - the number of stages in the pipeline or zero if the
               pipeline is broke.


--------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
/*  mvnppStageBeg
/*
/*  Put a local long program command in the given memory buffer.  The
/*  NULL that must follow commands is automatically included, and the
/*  number of words added to the buffer (2) is returned.  The
/*  mvnppLoadCmds routine calls this module.
/*
/*-----------------------------------------------------------------------*/

int mvnppStageBeg(memptr)
      USHORT *memptr;


---------------------------------------------------------------------------
```

```
/*------------------------------------------------------------------
/*  mvnppStageEnd
/*
/*  Put a null in the given memory buffer.  The value returned represents
/*  the number of short (16 bit) words written to the buffer, which
/*  will always be one.  The mvnppLoadCmds routine calls this module.
/*
/*-------------------------------------------------------------------*/

int mvnppStageEnd(memptr)
      USHORT *memptr;


      Return - number of short words written to the buffer (1)


------------------------------------------------------------------
```

```
/*------------------------------------------------------------------
/*  mvnppStageProg
/*
/*  Copy stage code from stage buffer to command buffer and add
/*  the control signals.  The DIV~ control signal indicating a valid
/*  data byte represents the high nibble and the stage code byte is the
/*  low nibble that forms the short word written to the buffer.
/*
/*-------------------------------------------------------------------*/

int mvnppStageProg(nbytes, stagecode, memptr)
      int nbytes
      char *stagecode
      USHORT *memptr

      Return - number of short words written to the buffer

      int nbytes - number bytes in the stage code block
      char *stagecode - buffer that holds the stage code
      USHORT *memptr  - buffer to hold stage command with control
                        signals

----------------------------------------------------------------
```

5.0 MVNPP Software Toplevel Routines

With the exception of 'mvnppLoadCmds', all 'toplevel' routines are called only when converting a stagecode .NOC (Neighborhood Object Code) file to the .LNOC format. The module 'makeLnoc' in the 'mvnpptest/makelnoc' directory is the main routine that does this. See the MVNPP Test Software Descriptions document (219800-13-T(1)) for details concerning 'makeLnoc'.

The 'mvnppLoadCmds' module calls several of the 'lowlevel' routines in order to set up a buffer that contains the stage commands and stagecode necessary to program the pipeline.

```
/*-------------------------------------------------------------------------
 *  mvnppAddStage
 *
 *  When a stage operation is read from the .NOC file (stored in long_op),
 *  it is compared with previously read-in operations.  If this new one is
 *  unique, it is added to a list of unique stage ops.  The long stage
 *  operation array must then be updated with the index of this operation.
 *  The index will either be the index of the operation that was matched
 *  (this new operation was not unique), or the index of the newly added
 *  stage operation to the list of unique ops.  A unique operation is one
 *  that has a different combination of a PRAM, NRAM and registers.
 *  This routine is called by mvnppCnvtOp().
 *
 *-----------------------------------------------------------------------*/

void mvnppAddStage(index, num_long_ops, long_op )
      int index;
      int *num_long_ops;
      struct long_stage_op *long_op;

      index - current index into the long stage op array
      num_long_ops - number of unique long stage ops
      long_op - new long stage op to be added to the array


 -------------------------------------------------------------------------
```

```
/*-------------------------------------------------------------------------
 *  mvnppCnvtOp
 *
 *  Read stagecode from a .NOC file and convert it to the long format.
 *
 *-----------------------------------------------------------------------*/

void mvnppCnvtOp(fp_in,fp_out)
      FILE *fp_in;
      FILE *fp_out;

      fp_in - file pointer to the source .NOC file
      fp_out - file pointer to the destination .LNOC file

 -------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
 * mvnppLoadCmds
 *
 * Load the memory (usually ROI memory) with the stage commands for a
 * programming cycle.  Return the size (in SHORTS) of the stage command
 * memory area.  The following routines are called:  mvnppFrameBeg,
 * mvnppStageBeg, mvnppStageProg, mvnppStageEnd, and mvnppFrameEnd.  First
 * a global activate command is put in the buffer, followed by a program
 * long stage command.  The actual stagecode follows (the valid data bit
 * is ORed in with each word), and at the end, 'pad' number of NULL commands.
 * Local deactivates are then sent so that any stages that have not been
 * programmed will not alter the image.  The end of the data in memory will
 * contain the global start of image command.
 *
 *----------------------------------------------------------------------*/

int mvnppLoadCmds(nstages, nactive, memptr, stagecode, codesizes, pad)
      int nstages;
      int nactive;
      USHORT *memptr;
      UCHAR *stagecode;
      int  *codesizes;

      nstages - the number of stages in the pipeline
      nactive - the number of stages in the pipeline to be programmed
      memptr - the address of the memory to be loaded with stage commands
      stagecode - the buffer of stage programs
      codesizes - the size of each stage program in stagecode

      Return: the number of words written to the address 'memptr'

 -----------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
 *  mvnppMakeStage
 *
 *  Make a long format stage structure from a PRAM stage op and an
 *  NRAM stage op structure.  This routine is called by mvnppCnvtOp().
 *
 *---------------------------------------------------------------------*/

void mvnppMakeStage(nram_op, pram_op, long_op)
      struct stage_op *nram_op;
      struct stage_op *pram_op;
      struct long_stage_op *long_op;

      nram_op - a c4pl XFORM or XFORM2 stage op with stage registers and
              an nram
      pram_op - a c4pl PRAM stage op
      long_op - a long format stage op with registers and pram and/or
              nram if necessary

 ------------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
 *  mvnppReadNOC
 *
 *  Read stagecode from a C4PL .NOC file and store the stage op numbers
 *  in an array, and also store the stage operations which are read in.
 *  Return the number of elements stored.
 *
 *---------------------------------------------------------------------*/

int mvnppReadNOC(fp)
      FILE *fp;

      fp - file pointer to a C4PL format .NOC file

      Return - the number of elements in the stage_op_array

 -----------------------------------------------------------------------
```

```
/*-----------------------------------------------------------------------
 *  mvnppWriteOps
 *
 *  Write stage operations converted to long format to the output file.
 *  This routine is called by mvnppCnvtOp().  The format of the file is
 *  as follows:
 *
 *                   | ---------------------------------- |
 *          0        | magic number - 0x0B                |
 *                   | ---------------------------------- |
 *          1        | dummy check sum                    |
 *                   | ---------------------------------- |
 *          2        | number of stages                   |
 *                   | ---------------------------------- |
 *          4        | offset to stageop 1                |
 *                   | ---------------------------------- |
 *          8        | offset to stageop 2                |
 *                   | ---------------------------------- |
 *                   |                .                   |
 *                   |                .                   |
 *       offset 1    | ---------------------------------- |
 *                   |  size lstage 1                     |
 *                   | ---------------------------------- |
 *                   |  lstage 1                          |
 *                   |                                    |
 *                   |                                    |
 *                   | ---------------------------------- |
 *                   |                .                   |
 *                   |                .                   |
 *       offset n    | ---------------------------------- |
 *                   |  size lstage n                     |
 *                   | ---------------------------------- |
 *                   |  lstage n                          |
 *                   |                                    |
 *                   |                                    |
 *                   | ---------------------------------- |
 *
 *---------------------------------------------------------------------*/

void mvnppWriteOps(fp, array_size, num_ops)
      FILE *fp;
      short array_size;
      int num_ops;

      fp - file pointer to .LNOC file
      array_size - number of elements in lstage_op_array
      num_ops - number of lstage_op elements

 -----------------------------------------------------------------------
```

6.0 ROI Math Utility

As described in the introduction, a new module has been added to the 'maxtools/roimath' directory that provides the 'roimath' utility from within a subroutine call. The current roimath routine is a standalone program that prompts for input about the configuration of the system being used - bus used, vertical and horizontal delays, etc. The output of the program lists values to be used when programming the ROI-STORE boards.

The new module added produces the same output, but it functions as a procedure call. All messages to the screen have been removed. The parameters and calling procedure is described below.

```
/*---------------------------------------------------------------------------
 *
 *  getROIparams
 *
 *  This routine calls the roimath utility that produces the eight ROI timing
 *  parameters.  There are five inputs to the routine, and the eight timing
 *  parameters are output.  The port_type is either P5_P6, or P6_P7, which
 *  are defined in mvnppbits.h.  This specifies which of the roistore ports
 *  will be used for the transfer.  When the MVNPP is located between the
 *  transmitting and receiving roistore boards, the horizontal and vertical
 *  delays should include the times returned from 'mvnppDelay'.
 *
 *--------------------------------------------------------------------------*/

void getROIparams(port_type,h_size,v_size,h_delay,v_delay,
                t_hstart,t_hend,t_vstart, t_vend,
                r_hstart,r_hend,r_vstart,r_vend)

  int port_type;  /* roistore ports to be used P5_P6 or P7_P8 are valid */
  int h_size;     /* horizontal image size
  int v_size;     /* vertical image size
  int h_delay;    /* horizontal delay introduced between xmit and rcv
                  /*   roistore boards - from MVNPP, MAX-MUX, etc.
  int v_delay;    /* vertical delay introduced between roistores

  int *t_hstart, *t_hend, *t_vstart, *t_vend;  /* returned xmit parameters */
  int *r_hstart, *r_hend, *r_vstart, *r_vend;  /* returned recv parameters */

 ---------------------------------------------------------------------------
```